SOFTWARE DESIGN DOCUMENT

for

Personal Library Management System

Library Hub

"Library Hub" serves as the product name for the Personal Library Management System, representing the centralized hub where users manage all their media collections in one unified platform.

Version	Issue Date	Issue Number
1.0 (Final)	December 14, 2025	SDD-PLMS-001

Prepared by

VisionSoft

Ali Emre Büyükersoy

Özlem Malkoç

Burak Altıparmak

Ceyda Özbey

Baran Şahin

CSE443 - Object-Oriented Analysis and Design

Gebze Technical University

Fall 2025

Table of Contents

1. INTRODUCTION

1.1 Purpose of the System

This Software Design Document (SDD) describes the architecture and system design of the Personal Library Management System (PLMS). It outlines the design goals, subsystem decomposition, hardware-software mapping, data management approach, access control mechanisms, and boundary conditions. This document serves as a blueprint for implementation and provides guidance for developers, testers, and maintainers of the system.

1.2 Design Goals

The PLMS design encompasses a cross-platform desktop application for managing personal media collections including books, movies, TV series, and music. The system architecture addresses:

- Desktop application for Windows, macOS, and Linux platforms
- Centralized data management with local database storage
- User authentication and profile management
- Media item CRUD operations with metadata enrichment
- Custom lists and favorites management
- Search and filtering capabilities
- Data import/export functionality
- Cross-device synchronization (future enhancement)

1.3 Definitions, Acronyms, and Abbreviations

Key technical terms and acronyms used throughout this document are defined in Section 5 (Glossary of Terms). The most frequently used abbreviations include:

- PLMS - Personal Library Management System
- MVC - Model-View-Controller architectural pattern
- SDD - Software Design Document
- RAD - Requirements Analysis Document
- UC - Use Case
- NFR - Non-Functional Requirement
- CRUD - Create, Read, Update, Delete
- UML - Unified Modeling Language
- API - Application Programming Interface

Note: Complete definitions are provided in Section 5 (Glossary of Terms).

1.4 References

- Requirements Analysis Document for PLMS (RAD-PLMS-001, Version 1.0, November 7, 2025)
- IEEE Std 1016-2009, IEEE Standard for Software Design Descriptions
- UML 2.5.1 Specification, Object Management Group
- Design Patterns: Elements of Reusable Object-Oriented Software, Gamma et al.
- CSE443 Course Materials, Object-Oriented Analysis and Design, Fall 2025

1.5 Overview

This Software Design Document is organized into six main sections:

- Section 1 (Introduction) describes the document purpose, system scope, design goals, key definitions, and references.

2. CURRENT SOFTWARE ARCHITECTURE

- Section 3 (Proposed Software Architecture) presents the three-tier MVC architecture with detailed subsystem decomposition, hardware/software mapping, data management, security mechanisms, control flow, and boundary conditions. UML diagrams illustrate class structure, deployment, and package organization.
- Section 4 (Subsystem Services) enumerates services provided by each subsystem with use case mappings.
- Section 5 (Glossary of Terms) defines technical terms and acronyms.
- Section 6 (Traceability) establishes requirement-to-design mappings and documents design patterns.

Currently, personal library management relies on manual methods such as physical notebooks, spreadsheet applications (Excel, Google Sheets), or disconnected mobile apps. Users maintaining media collections face significant challenges with these ad-hoc approaches.

2.2 Limitations of Current Approach

Lack of Integration: Different media types require separate tracking systems. No unified collection view. Manual data entry from multiple sources. No automated metadata enrichment.

Poor Data Management: Spreadsheets lack referential integrity. Manual entry causes inconsistent formats and typos. No validation rules for domain-specific data (ISBN, dates). Duplicate detection is manual.

Limited Functionality: Basic filtering and sorting only. No full-text search or fuzzy matching. Cannot track reading progress or viewing history. Limited categorization options.

Synchronization Issues: Local files cannot sync across devices. Cloud spreadsheets require constant internet. Version conflicts when edited on multiple devices. No real-time collaboration.

These limitations demonstrate the clear need for a dedicated personal library management solution that automates metadata enrichment, ensures data quality, provides advanced search, and manages data persistence effectively.

3. PROPOSED SOFTWARE ARCHITECTURE

3.1 Overview

The PLMS employs a three-tier Model-View-Controller (MVC) architecture as specified in NFR-27 of the Requirements Analysis Document. This architectural pattern ensures clear separation of concerns between user interface, business logic, and data management components.

The system follows object-oriented design principles (NFR-26) with modular subsystems that can be developed, tested, and maintained independently. The architecture supports the cross-platform desktop application requirement (NFR-36, NFR-38) deployable on Windows, macOS, and Linux.

3.2 Subsystem Decomposition

The system is decomposed into six major subsystems based on functional cohesion and the use cases defined in the Requirements Analysis Document:

Subsystem	Responsibilities	Related Use Cases
User Management	Authentication, registration, profile management, session handling	UC-001, UC-002, UC-003, UC-004, UC-005
Library Management	Media item CRUD, search, filtering, metadata enrichment	UC-006, UC-007, UC-008, UC-009, UC-010
List Management	Custom lists, favorites, reading/watching lists	UC-011, UC-012, UC-013
Data Import/Export	CSV/JSON import, data export, batch operations	UC-014
Search & Filter	Full-text search, advanced filtering, sorting	UC-015
Synchronization	Cross-device data sync, conflict resolution	UC-016 (Future)

3.2.1 User Management Subsystem

Components:

- UserController: Handles user registration, login, logout, profile operations
- AuthenticationService: Validates credentials, manages sessions
- UserRepository: Database operations for user data

- PasswordManager: Secure password hashing and validation

3.2.2 Library Management Subsystem

Components:

- MediaItemController: Handles item CRUD operations
- MediaItemService: Business logic for item management
- MetadataEnricher: Fetches metadata from external APIs (Google Books, OMDB)
- MediaItemRepository: Database persistence for media items
- MediaItemFactory: Creates type-specific item instances (Book, Movie, TVSeries, Music)

3.2.3 List Management Subsystem

Components:

- ListController: Manages list operations
- ListService: List creation, item addition/removal, reordering
- ListRepository: Database operations for lists and list items

3.2.4 Search & Filter Subsystem

Components:

- SearchController: Handles search requests
- SearchService: Full-text search implementation
- FilterBuilder: Constructs dynamic filter queries
- SortManager: Handles sorting by various criteria

3.2.5 Data Import/Export Subsystem

Components:

- ImportExportController: Handles file upload/download
- CSVProcessor: Parses and validates CSV files
- JSONProcessor: Handles JSON data format
- DataValidator: Validates imported data against schema

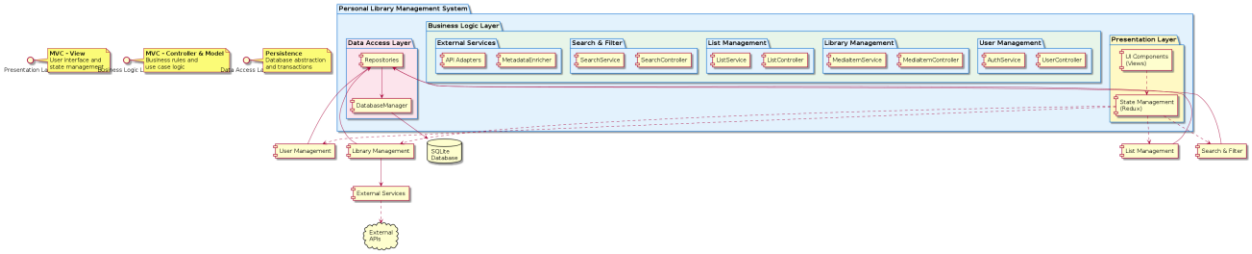
3.2.6 Data Access Layer

Components:

- DatabaseManager: Manages database connections and transactions
- RepositoryBase: Abstract base class for all repositories
- QueryBuilder: Constructs SQL queries programmatically
- MigrationManager: Handles database schema updates

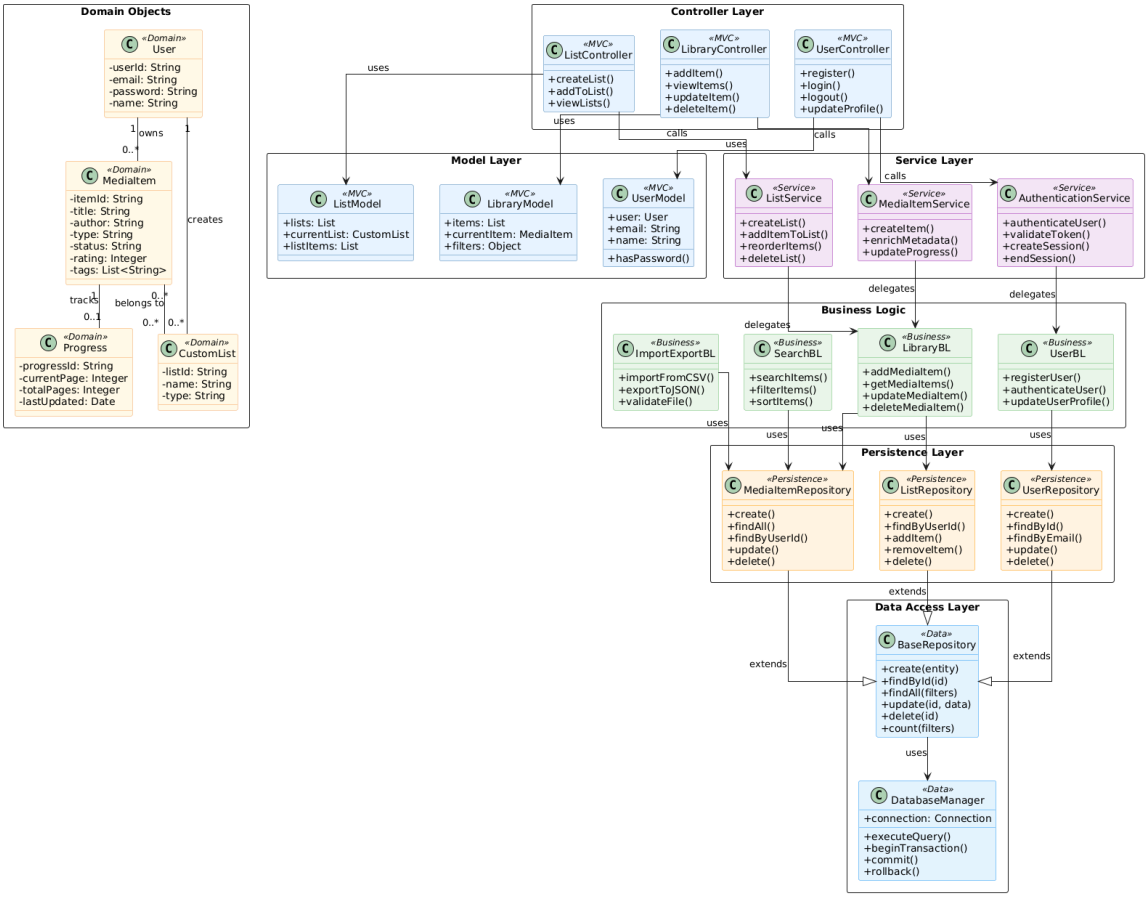
3.2.7 Subsystem Package Organization

PLMS Package Diagram



The package diagram shows the modular organization of subsystems and their dependencies. The three-tier architecture is clearly visible: Presentation Layer (UI components and state management), Business Logic Layer (six subsystems with specific responsibilities), and Data Access Layer (repositories and database management). Dependencies flow downward, ensuring proper separation of concerns per MVC pattern.

PLMS Class Diagram



The class diagram above illustrates the object-oriented structure of PLMS following MVC pattern. Model classes (User, MediaItem hierarchy, UserList) represent domain entities. Controller classes handle user requests. Service classes implement business logic. Repository classes provide data access abstraction over SQLite database.

3.3 Hardware/Software Mapping

3.3.1 Development Technology Stack

Based on the cross-platform desktop requirement (NFR-38), the system uses the following technologies:

Desktop Framework: Electron 28.x for cross-platform desktop development

Programming Language: JavaScript with TypeScript 5.x for type safety

User Interface: React 18.x with Redux Toolkit for state management

Database: SQLite for local storage (embedded, no server required)

ORM/Database Access: Sequelize ORM for Node.js/TypeScript with SQLite

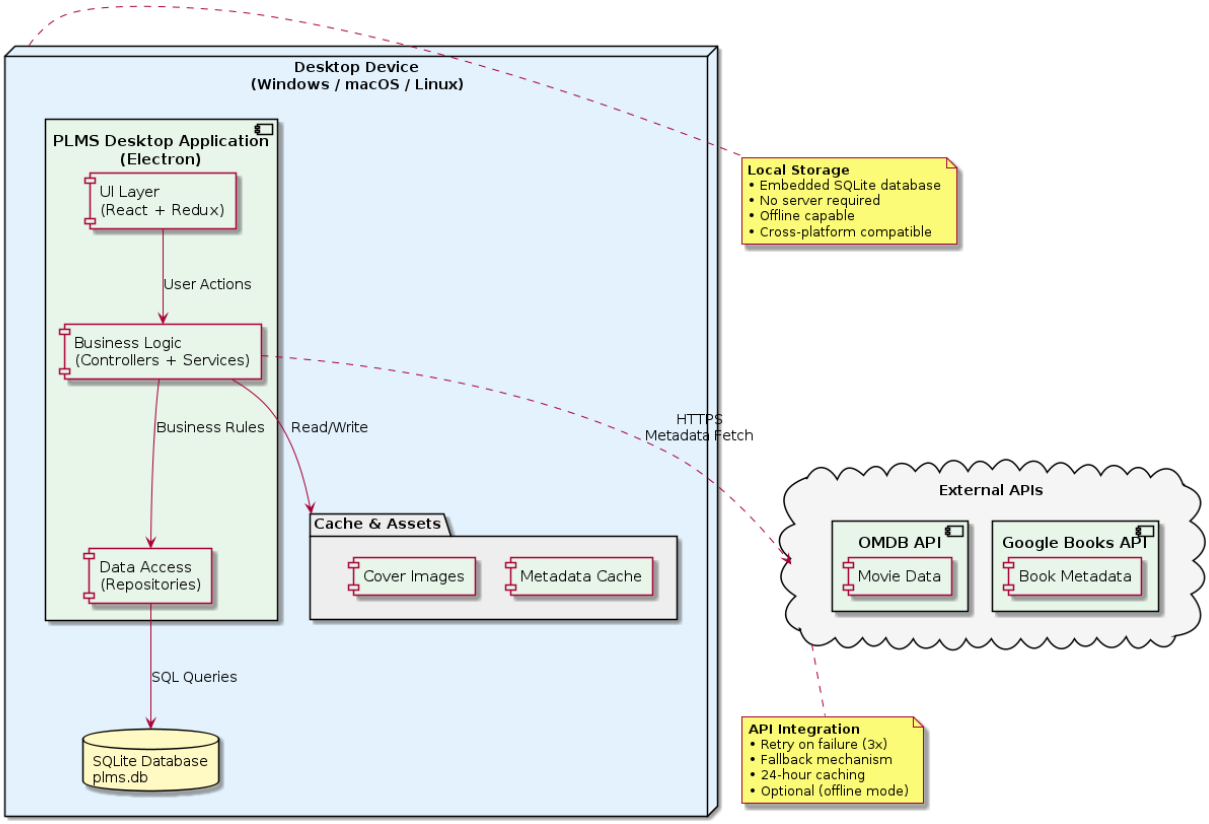
External API Integration: Axios/Fetch for HTTP requests to metadata services

3.3.2 Deployment Architecture

The system is packaged as a standalone desktop application (NFR-36) with the following deployment model:

- Single-user desktop application with local database
- SQLite database file stored in user application data directory
- Application installer (<50MB per NFR-39) includes all dependencies
- Automatic dependency checking during installation (NFR-40)
- Platform-specific installers: .exe (Windows), .dmg (macOS), .deb/.rpm (Linux)

PLMS Deployment Architecture



The deployment diagram shows the physical architecture. The desktop application (Electron framework) contains presentation, business logic, and data access layers. SQLite database provides local persistent storage. External APIs (Google Books, OMDB) are accessed over HTTPS for metadata enrichment with retry logic and caching.

3.3.3 Client Requirements

- Operating System: Windows 10+, macOS 10.15+, Ubuntu 20.04+

- RAM: 4GB minimum, 8GB recommended
- Storage: 500MB for application, 1GB+ for user data
- Display: 1280x720 minimum resolution
- Internet: Required for metadata enrichment, optional for offline use

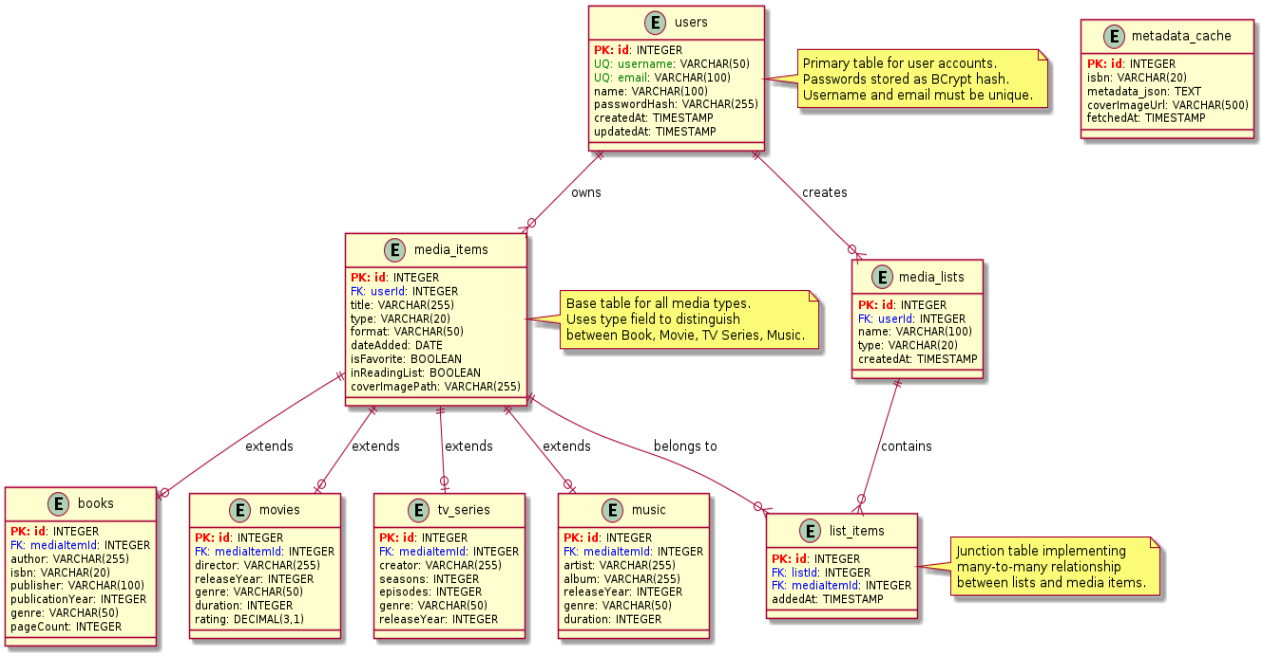
3.4 Persistent Data Management

3.4.1 Database Design

The system uses SQLite as the embedded database engine, storing data locally on the user device. The database schema follows third normal form (3NF) to minimize redundancy while maintaining query performance.

Core Database Tables:
users: Stores user account information including credentials and profile data
media_items: Base table for all media with common attributes (title, type, status, dates)
books: Book-specific attributes (author, ISBN, publisher, pages)
movies: Movie-specific attributes (director, runtime, IMDb ID)
tv_series: TV series attributes (seasons, episodes, network)
music: Music attributes (artist, album, genre, duration)
user_lists: User-created lists (name, type, creation date)
list_items: Many-to-many relationship between lists and media items
metadata_cache: Cached metadata from external APIs to reduce repeated requests
progress_logs: Reading/viewing progress tracking

PLMS ER Diagram



3.4.2 Data Persistence Strategy

- SQLite Write-Ahead Logging (WAL) mode for concurrent read access
- Foreign key constraints enforce referential integrity
- Indexes on frequently queried columns (user_id, media_type, title)
- Full-text search indexes for title and author fields
- Transaction management ensures ACID properties
- Automatic backup on application shutdown
- Database file encryption for sensitive user data (optional)

3.4.3 Backup and Recovery

- Automatic daily backups to user-specified location
- Manual backup option in application settings
- Export functionality creates CSV/JSON backups (NFR-44)
- Backup files include timestamp for version control
- Restore functionality validates backup before restoration

3.5 Access Control and Security

3.5.1 Authentication

The system implements secure user authentication as required by use cases UC-001 and UC-002:

- Password hashing using bcrypt algorithm with per-user salts
- Session management with secure session tokens
- Password strength requirements (minimum 8 characters, mixed case, numbers)
- Account lockout after 5 failed login attempts
- Password reset via email-based password reset (UC-003)
- Session timeout after 30 minutes of inactivity

3.5.2 Access Control Matrix

The system implements role-based access control (RBAC) with the following access matrix defining permissions for different user roles and system resources:

Resource / Operation	Anonymous	Authenticated User	Resource Owner	System Admin
User Registration	✓	—	—	✓
User Login	✓	—	—	✓
View Own Profile	—	✓	✓	✓
Manage Own Library	—	—	✓	✓

Create/Edit Lists	—	—	✓	✓
Delete Media Items	—	—	✓	✓
System Configuration	—	—	—	✓

- ✓ = Permission granted
- — = Permission denied
- Anonymous: Unauthenticated visitors (can only register/login)
- Authenticated User: Logged-in user (can view own profile)
- Resource Owner: User who created the resource (full CRUD on own data)
- System Admin: Reserved for future multi-user features

Access Control Legend:

3.5.3 Data Security

- Local database encryption using SQLCipher (optional)
- Secure credential storage in OS keychain
- Input validation to prevent SQL injection
- HTTPS for all external API communications
- User data privacy per GDPR requirements (NFR-41, NFR-42, NFR-43)

3.6 Global Software Control

3.6.1 MVC Architecture Implementation

Following NFR-27, the system implements Model-View-Controller pattern with clear separation:

Model Layer: Represents data structures (User, MediaItem, List) and business logic. Manages database interactions through repositories. Independent of UI implementation.

View Layer: User interface components displaying data to users. Observes model changes and updates display. Sends user actions to controllers. Platform-specific UI frameworks (React).

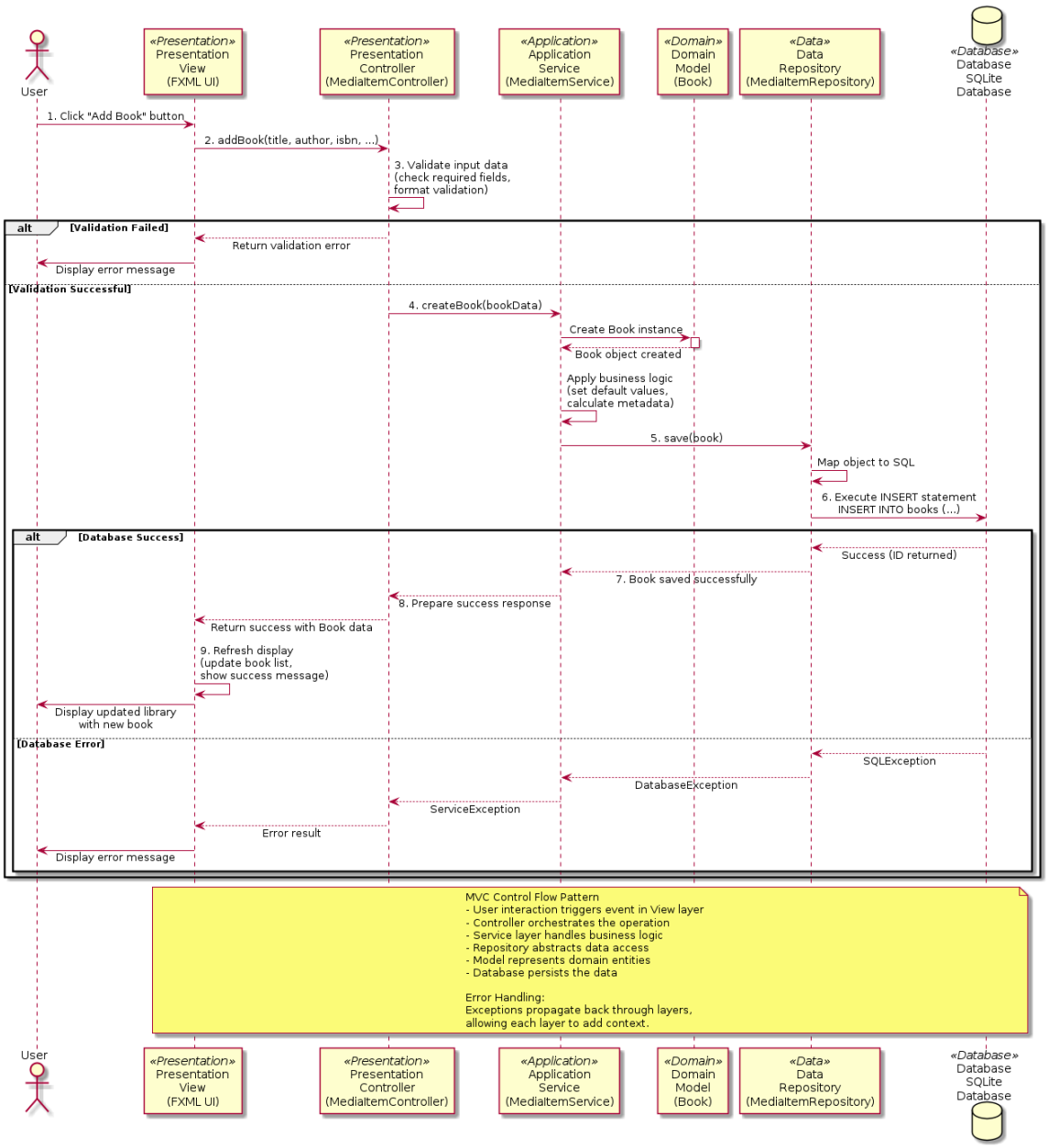
Controller Layer: Receives user input from views. Processes requests and updates models. Coordinates between views and models. Implements use case logic.

3.6.2 Control Flow

Typical user operation control flow:

- 1. User interacts with View (e.g., clicks "Add Book" button)
- 2. View invokes Controller method (e.g., MediaItemController.addBook())
- 3. Controller validates input data
- 4. Controller calls Service layer for business logic (MediaItemService.createBook())
- 5. Service updates Model and persists through Repository (MediaItemRepository.save())
- 6. Repository executes database operations
- 7. Success/failure result propagates back to Controller
- 8. Controller updates View with result
- 9. View refreshes display to show updated state

Plms Control Flow Sequence



3.6.3 Transaction Management

- Database transactions ensure ACID properties for critical operations
- Begin transaction before multiple related database updates
- Commit transaction on successful completion
- Rollback transaction on any error to maintain consistency
- Transaction scope: Service layer methods that modify multiple entities

3.6.4 Concurrency Control

As a single-user desktop application with local SQLite database, PLMS has minimal concurrency concerns compared to multi-user systems. However, the following synchronization and concurrency issues are addressed:

Database Access Concurrency: SQLite supports concurrent reads but serializes writes. The application uses Write-Ahead Logging (WAL) mode to allow simultaneous read operations while a write is in progress. Transaction isolation level is set to READ COMMITTED to prevent dirty reads.

UI Thread vs Background Operations: Metadata enrichment and external API calls are performed asynchronously on background threads to prevent UI freezing. The main UI thread communicates with worker threads via message queues and callback mechanisms. Redux state updates are dispatched from background threads using thread-safe actions.

File System Operations: Cover image downloads and cache operations use file locking mechanisms to prevent corruption. Temporary files are written with unique names and atomically renamed upon completion to ensure consistency.

State Management Concurrency: Redux store provides thread-safe state updates through its serialized action dispatch mechanism. All state modifications go through reducers, ensuring predictable state transitions even with concurrent user actions.

Future Multi-Device Sync Considerations (Future Enhancement): Future Multi-Device Sync Considerations: When synchronization is implemented (UC-016), vector clocks will track causality between operations on different devices. Conflicts will be detected by comparing version vectors and resolved using last-write-wins strategy with timestamp comparison. This mechanism is considered as a future enhancement and is not part of the initial implementation.

3.7 Boundary Conditions

3.7.1 System Initialization

- Check and create application data directory if not exists
- Initialize SQLite database file if first run
- Run database schema migrations if needed
- Load application configuration from settings file
- Verify database connectivity and integrity
- Initialize UI framework and display login screen

3.7.2 System Shutdown

- Save any pending user changes to database
- Close all database connections properly

- Create automatic backup if configured
- Save application state (window size, last view)
- Clear sensitive data from memory
- Gracefully terminate application processes

3.7.3 Error Handling

The system implements robust error handling as per NFR-8:

- Try-catch blocks around all database operations
- User-friendly error messages displayed in UI
- Detailed error logging to application log file
- Graceful degradation: continue operation when possible
- Specific handling for common errors (duplicate entries, invalid input)
- Network error handling for metadata fetching (NFR-27a, NFR-27b)

3.7.4 Failure Scenarios

Database Corruption: Detect on startup → Attempt automatic repair → Restore from last backup → Notify user

Network Unavailable: Detect when metadata fetch fails → Queue request for retry → Continue offline mode → Sync when network restored (NFR-27c, NFR-27d, NFR-27e)

Insufficient Storage: Check available space before operations → Warn user if below 100MB → Prevent new additions → Suggest cleanup or backup

Corrupted Import File: Validate file format → Parse with error recovery → Report invalid rows → Import valid data only

4. SUBSYSTEM SERVICES

This section describes the services provided by each subsystem, mapping to the use cases defined in the Requirements Analysis Document.

4.1 User Management Services

Provides authentication and user profile services:

- registerUser(email, password, name) → Creates new user account (UC-001)
- authenticateUser(email, password) → Validates credentials and creates session (UC-002)
- logoutUser(userId) → Ends user session (UC-004)
- requestPasswordReset(email) → Initiates password recovery (UC-003)
- resetPassword(token, newPassword) → Completes password reset
- updateUserProfile(userId, profileData) → Updates user information (UC-005)
- deleteAccount(userId) → Removes user account and associated data

4.2 Library Management Services

Core media management operations:

- addMediaItem(userId, itemData) → Creates new media item (UC-006)
- getMediaItems(userId, filters) → Retrieves user library with optional filtering (UC-007)
- getMediaItemDetails(itemId) → Retrieves full item information
- updateMediaItem(itemId, updates) → Modifies item data (UC-009)
- deleteMediaItem(itemId) → Removes item from library (UC-010)
- enrichMetadata(itemId) → Fetches external metadata (books: Google Books, movies: OMDB)
- updateProgress(itemId, progress) → Records reading/viewing progress
- getProgressHistory(itemId) → Retrieves progress timeline

4.3 List Management Services

Custom list operations:

- createList(userId, listName, listType) → Creates new list (UC-012)
- addItemToList(listId, itemId) → Adds item to list (UC-011)
- removeItemFromList(listId, itemId) → Removes item from list
- getUserLists(userId) → Retrieves all user lists (UC-013)
- getListItems(listId) → Retrieves items in specific list
- reorderListItems(listId, itemPositions) → Changes item order in list
- deleteList(listId) → Removes list (items remain in library)

4.4 Search & Filter Services

Advanced search and filtering:

- searchItems(userId, searchText) → Full-text search across title, author, tags (UC-008)
- filterItems(userId, filterCriteria) → Filter by type, status, genre, year (UC-015)
- sortItems(items, sortField, direction) → Sort results by various criteria
- getFacets(userId) → Returns available filter options with counts
- getSuggestions(userId, partialText) → Autocomplete for search queries

4.5 Data Import/Export Services

Data portability services:

- importFromCSV(userId, filePath) → Imports media items from CSV file (UC-014)
- importFromJSON(userId, filePath) → Imports from JSON format
- exportToCSV(userId, filePath) → Exports user library to CSV
- exportToJSON(userId, filePath) → Exports to JSON format
- validateImportFile(filePath) → Checks file format before import
- generateImportReport(importResults) → Creates summary of import operation

4.6 Data Access Layer

Generic database operations (used by all subsystems):

- create(entity) → Inserts new record
- findById(id) → Retrieves record by primary key
- findAll(filters) → Retrieves multiple records with optional filtering
- update(id, data) → Modifies existing record
- delete(id) → Removes record
- count(filters) → Counts records matching criteria
- beginTransaction() → Starts database transaction
- commit() → Commits transaction
- rollback() → Reverts transaction changes

5. GLOSSARY OF TERMS

This glossary defines technical terms and acronyms used throughout this document.

ACID: Atomicity, Consistency, Isolation, Durability - database transaction properties

API: Application Programming Interface - protocols for software communication

CRUD: Create, Read, Update, Delete - basic data operations

CSV: Comma-Separated Values - plain text file format for tabular data

DAO: Data Access Object - design pattern for abstracting database access

GDPR: General Data Protection Regulation - EU data protection law

IMDb: Internet Movie Database - online movie and TV database

ISBN: International Standard Book Number - unique book identifier

JSON: JavaScript Object Notation - lightweight data interchange format

MVC: Model-View-Controller - architectural pattern separating concerns

NFR: Non-Functional Requirement - quality attribute requirement

OOP: Object-Oriented Programming - programming paradigm based on objects

ORM: Object-Relational Mapping - technique for database abstraction

PLMS: Personal Library Management System - the system described in this document

RAD: Requirements Analysis Document - specification of system requirements

SDD: Software Design Document - this document

SQL: Structured Query Language - language for database management

SQLite: Self-contained, serverless SQL database engine

UC: Use Case - functional requirement describing system behavior

UI: User Interface - means of user interaction with software

UML: Unified Modeling Language - standard for software modeling

WAL: Write-Ahead Logging - database logging method for durability

6. TRACEABILITY

6.1 Requirements to Design Mapping

This section establishes traceability between use cases in RAD-PLMS-001 and design components in this SDD.

Use Case ID	Use Case Name	Design Component	Subsystem
UC-001	User Registration	UserController.registerUser()	User Management
UC-002	User Login	UserController.authenticateUser()	User Management
UC-003	Forgot Password	UserController.resetPassword()	User Management
UC-004	User Logout	UserController.logoutUser()	User Management
UC-005	Update Profile	UserController.updateUserProfile()	User Management
UC-006	Add Media Item	MediaItemController.addMediaItem()	Library Management
UC-007	View Library	MediaItemController.getMediaItems()	Library Management
UC-008	Search Items	SearchController.searchItems()	Search & Filter
UC-009	Update Media Item	MediaItemController.updateMediaItem()	Library Management
UC-010	Delete Media Item	MediaItemController.deleteMediaItem()	Library Management
UC-011	Add to Favorites	ListController.addItemToList()	List Management
UC-012	Add to Reading List	ListController.createList()	List Management
UC-013	View Lists	ListController.getUserLists()	List Management
UC-014	Import/Export Data	ImportExportController	Data Import/Export
UC-015	Filter Items	SearchController.filterItems()	Search & Filter
UC-016	Sync Across Devices	SyncController (Future)	Synchronization

6.2 Non-Functional Requirements Mapping

Key NFRs addressed in the design:

- NFR-26 (OOP Principles) → Object-oriented architecture with classes and inheritance
- NFR-27 (MVC Pattern) → Clear MVC separation described in Section 3.6
- NFR-27a-e (Metadata Reliability) → Retry logic and fallback in MetadataEnricher
- NFR-36 (Standalone Application) → Desktop app with embedded database
- NFR-38 (Cross-platform) → Electron deployment architecture
- NFR-39 (Installer Size <50MB) → Minimal dependency packaging
- NFR-40 (Dependency Check) → Installation validation in deployment
- NFR-41-44 (Privacy/GDPR) → Data security measures and export capability

6.3 Design Patterns Used

The following design patterns support system requirements:

MVC (Model-View-Controller): Separates UI, business logic, and data - Addresses NFR-27 (Location: Overall architecture)

Repository Pattern: Abstracts data access and enables database independence (Location: Data Access Layer)

Factory Pattern: Creates type-specific media items (Book, Movie, TV, Music) (Location: MediaItemFactory)

Singleton Pattern: Ensures single database connection manager instance (Location: DatabaseManager)

Observer Pattern: Updates UI when model data changes (Location: View-Model communication)

Strategy Pattern: Pluggable search/filter algorithms (Location: Search & Filter subsystem)

6.4 Future Enhancements

The architecture supports these potential future enhancements without major redesign:

- UC-016: Cross-device synchronization with conflict resolution
- Mobile companion app (iOS/Android) sharing same database schema
- Cloud backup integration (optional user feature)
- Social features: sharing lists, recommendations
- Advanced analytics: reading statistics, trends
- Barcode scanner integration for easy book addition
- E-reader integration (Kindle, Kobo) for progress sync
- Additional media types (podcasts, audiobooks, magazines)
- Multi-user support with shared family libraries

6.5 Conclusion

This Software Design Document provides a comprehensive architectural blueprint for the Personal Library Management System that directly addresses all requirements specified in RAD-PLMS-001. The MVC architecture ensures clear separation of concerns, the subsystem decomposition enables modular development, and the design patterns promote code reusability and maintainability.

The design successfully balances functional requirements (16 use cases fully mapped to components) with non-functional requirements (performance, reliability, usability, cross-platform support). The SQLite-based local storage approach provides offline capability while maintaining simplicity for a desktop application.

All architectural decisions align with the object-oriented principles and MVC pattern specified in NFR-26 and NFR-27. The modular subsystem structure allows parallel development by team members and facilitates future enhancements without requiring core architecture changes.